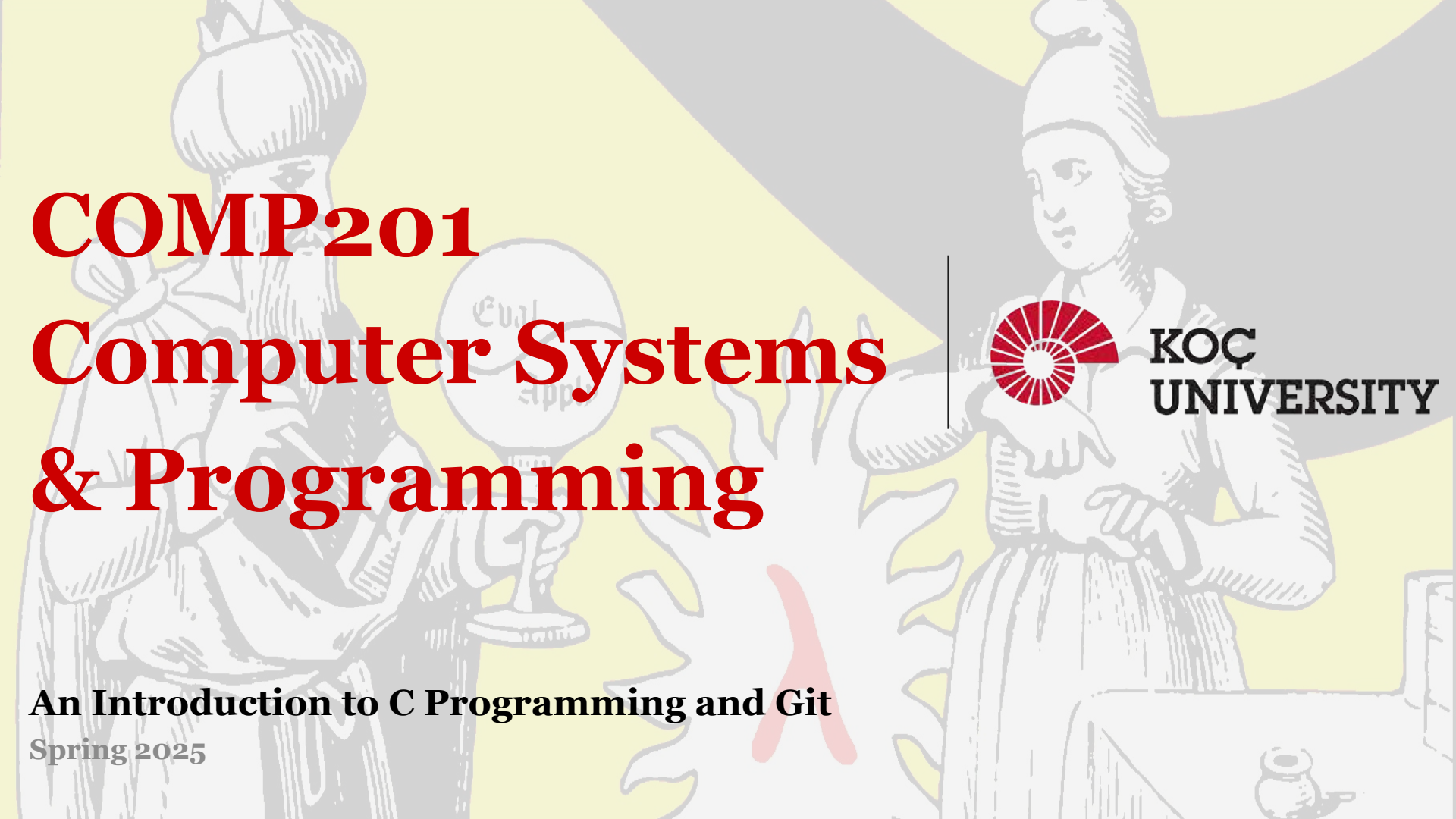


COMP201 Computer Systems & Programming

KOÇ UNIVERSITY

An Introduction to C Programming and Git

Spring 2025

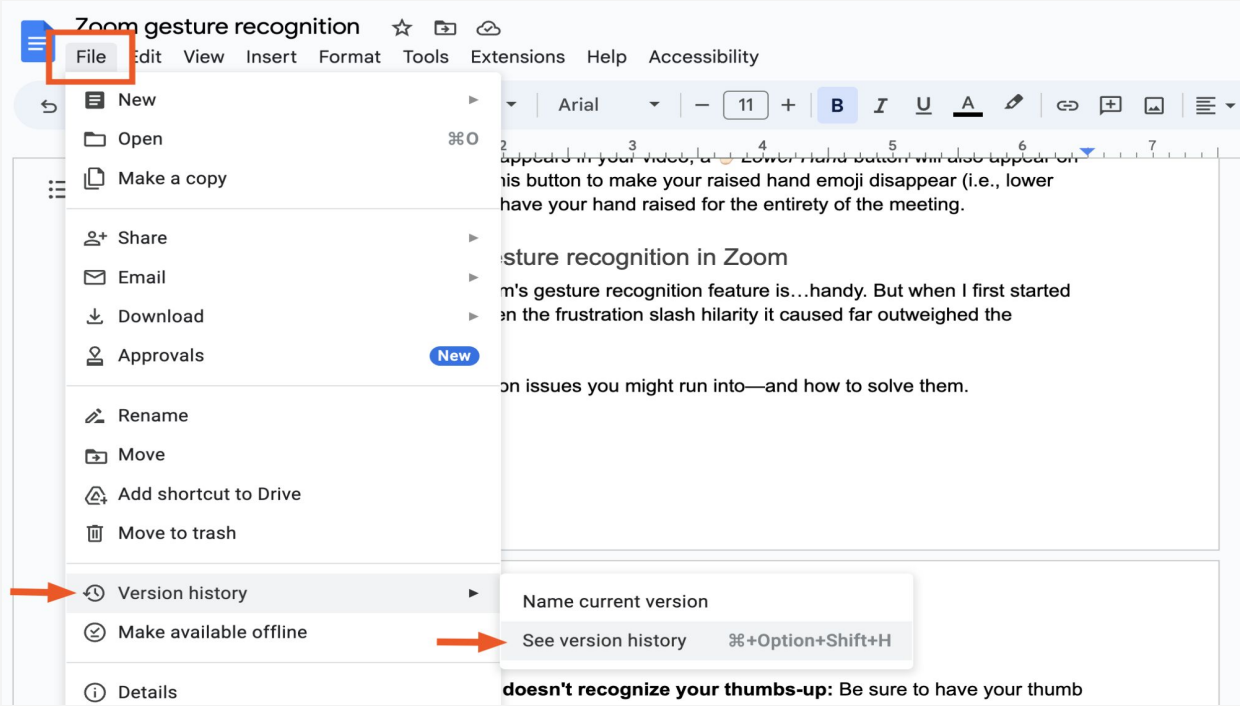
Objectives

- Understand version control systems (VCS) and Git
- GitHub Classroom
- Learn how to use basic Git commands
- Get familiar with LinuxPool machines
- Create bash scripts
- Get familiar with basic C programming concepts

Version Control Systems (VCS)

- Many files are getting created as we work on our projects.
- Eventually, we need a structured way to keep track different versions of our files.
- A version control system (VCS) is a piece of software which manages different versions of your files and folders for you.
- A good VCS will let you look at old versions of files and restore files (or information) which you might have accidentally deleted.
- **We have already seen them !**

Version Control Systems (VCS)



KOÇ UNIVERSITY

Version Control Systems (VCS)

- Google Docs automatically tracks file history, providing version control.

- Zipping project directories with versioned filenames can also serve as a a basic VCS.

- However, we need more robust version control systems (VCS), which enable us to:
  - track changes to a file without duplicating entire directories.
  - revert specific files or parts of files to previous versions.
  - maintain a clear, structured order of changes over time.
  - facilitate seamless collaboration.
  - experiment with different copies of existing files without manipulating actual files.

Version Control Systems (VCS)

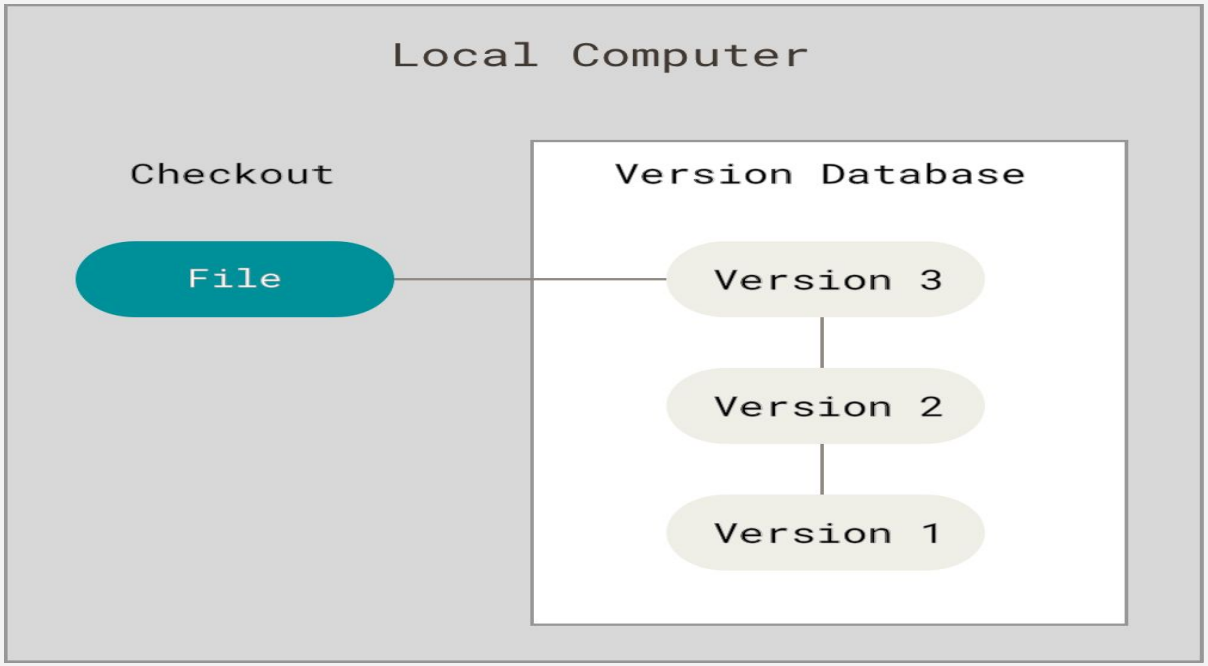


Figure 1: Local version control

Version Control Systems (VCS)

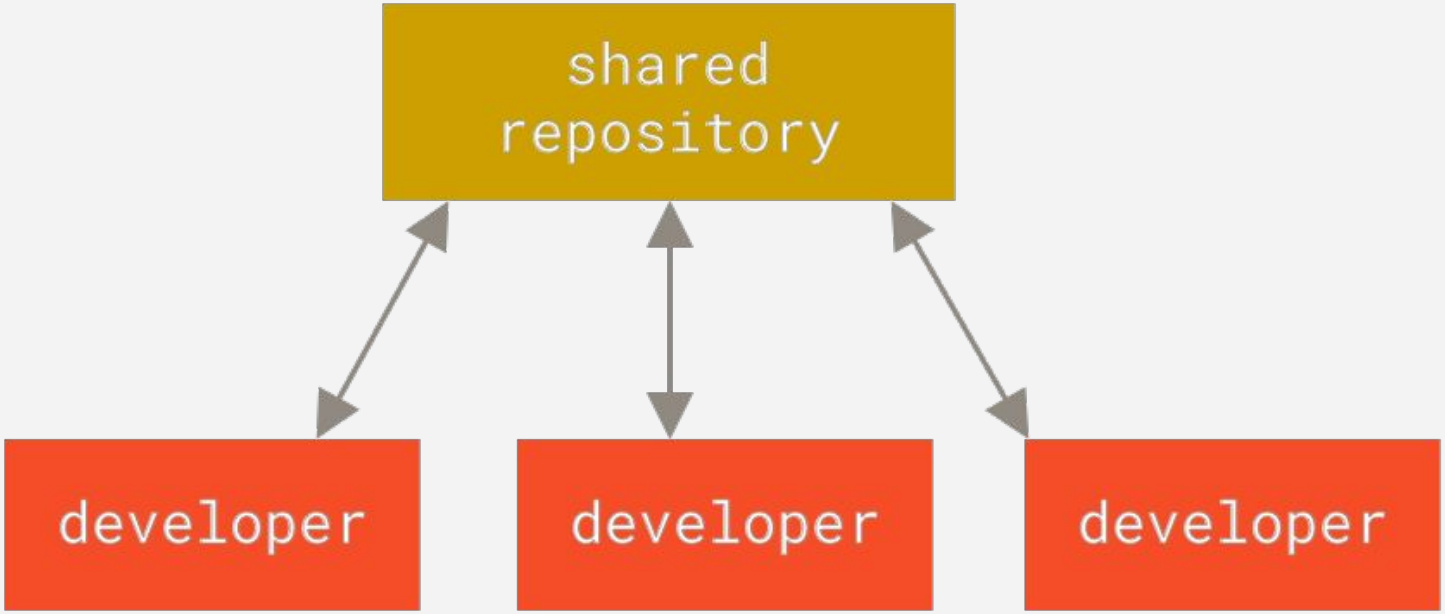


Figure 2: Centralized version control

Version Control Systems (VCS)

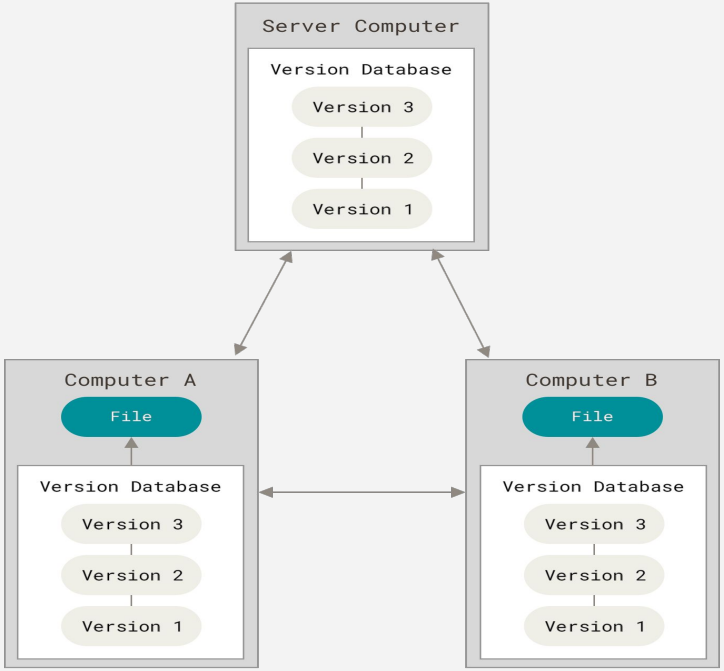


Figure 3: Distributed version control

git

- A free and open-source distributed version control system (DVCS) developed by **Linus Torvalds**, the creator of Linux.
- You “clone” the central repository and “pull” changes from it.
- Your local repository is a complete copy of everything on the central remote repository.
- When you’re ready, you “push” changes back to the server.

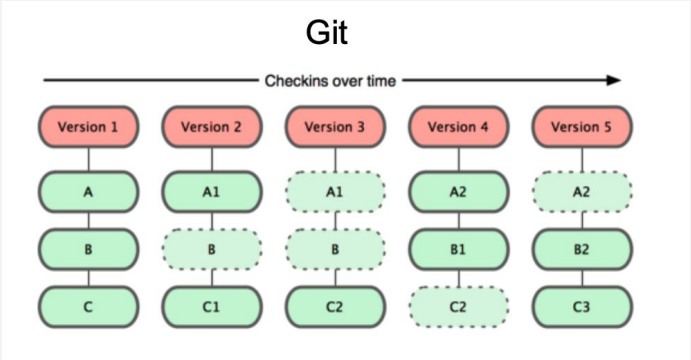


Figure 4: `git` checkins over time

Git website: https://git-scm.com/
Git for Computer Scientists: https://eagain.net/articles/git-for-computer-scientists/

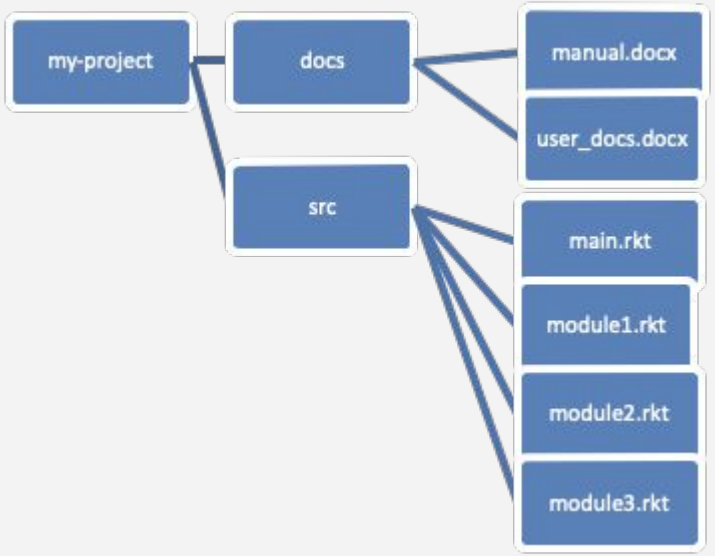
git

- In git-speak, a “version” is called a “commit”.

- Git keeps track of the history of your commits, so you can go back and look at earlier versions, or just give up on the current version and go back some earlier version.

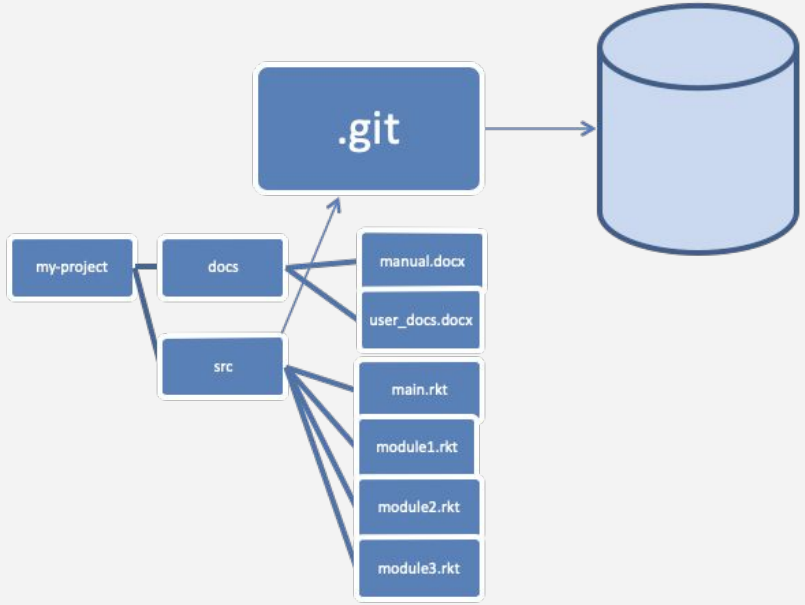
git

Here are your files, sitting in a directory called my-project



git

- When you have a git repository, you have an additional directory called .git, which points at a mini-filesystem.
- This file system keeps all your data, plus the bells and whistles that git needs to do its job.
- All this sits on your local machine.



`git` - clone

- You can clone your repository to create a local copy on your computer and sync between the two locations.

- Example command:

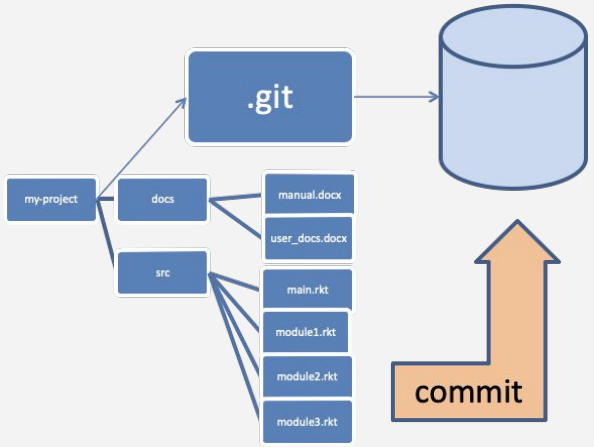
```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

- Cloning a repository.

KOÇ UNIVERSITY

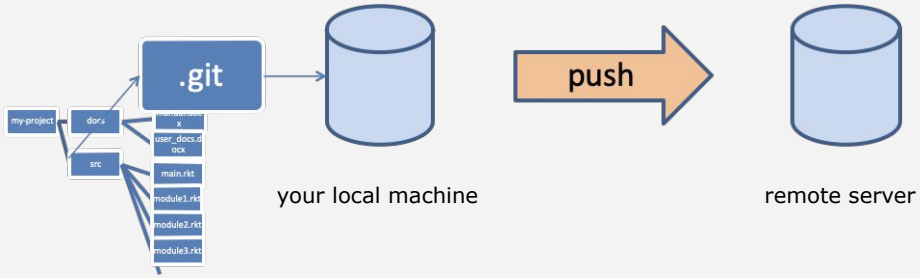
git - commit

- You edit your local files directly. Now, your mini-filesystem pointed by .git is behind.
- When you do a “commit”, you record all your local changes into the mini-fs.
- The mini-fs is “append-only”. Nothing is ever over-written there, so everything you ever commit can be recovered.



git - push

- At the end of each work session, you need to save your changes on the server. This is called a “push”. Now all your data is backed up.
- You can retrieve it, on your machine or some other machine.
- We can retrieve it (that’s how we collect assignments).

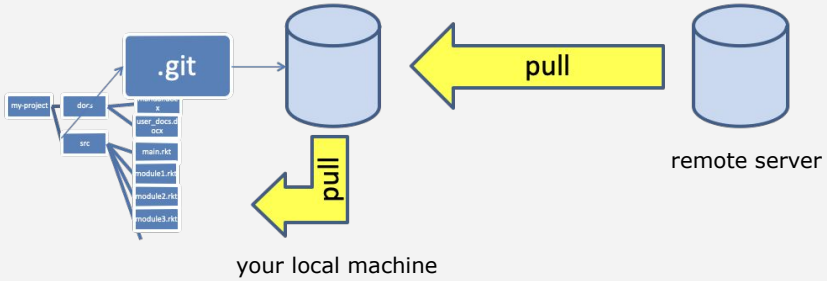


KOÇ UNIVERSITY

git - pull

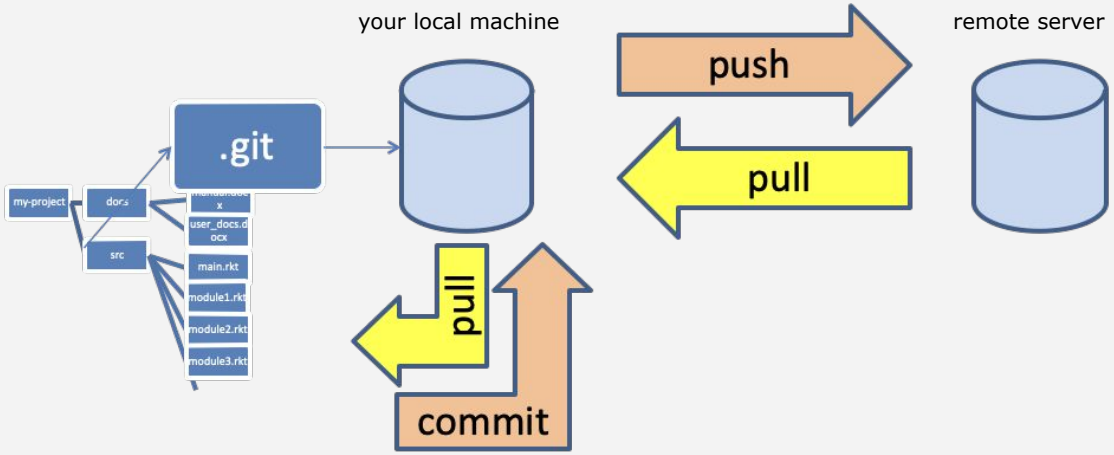
- To retrieve your data from the server, you do a “pull”. A “pull” takes the data from the server and puts it both in your local mini-fs and in your ordinary files.

- If your local file has changed, git will merge the changes if possible. If it can’t figure out how to the merge, you will get an error message. We'll learn how to deal with these in the next lesson.



KOÇ UNIVERSITY

git - push & pull



`git` - Useful links

- `git` branch management
- `git` – Tutorial
- `git` - Cheatsheet
- `git` - Adding a new SSH key to you GitHub account

GitHub Classroom

- It is used to create and manage digital classrooms and assignments.
- You will get an invitation link to accept assignments.
- You will clone the template repository, work on it, and push changes back to your GitHub Classroom repository.
- *Assignment 0: Getting Started with Unix and C* aims at making sure you familiarize yourself with all the essential stuff.

C – Introduction

- Developed by AT&T Bell Labs in early 1970s. UNIX also developed at Bell Labs.
- The foundation of many modern languages (C++, Python, Java, etc.).
- Widely used in systems programming, embedded systems, game development, and more.
- Efficient, low-level access to memory, speed, and portability.

C – Development Environment

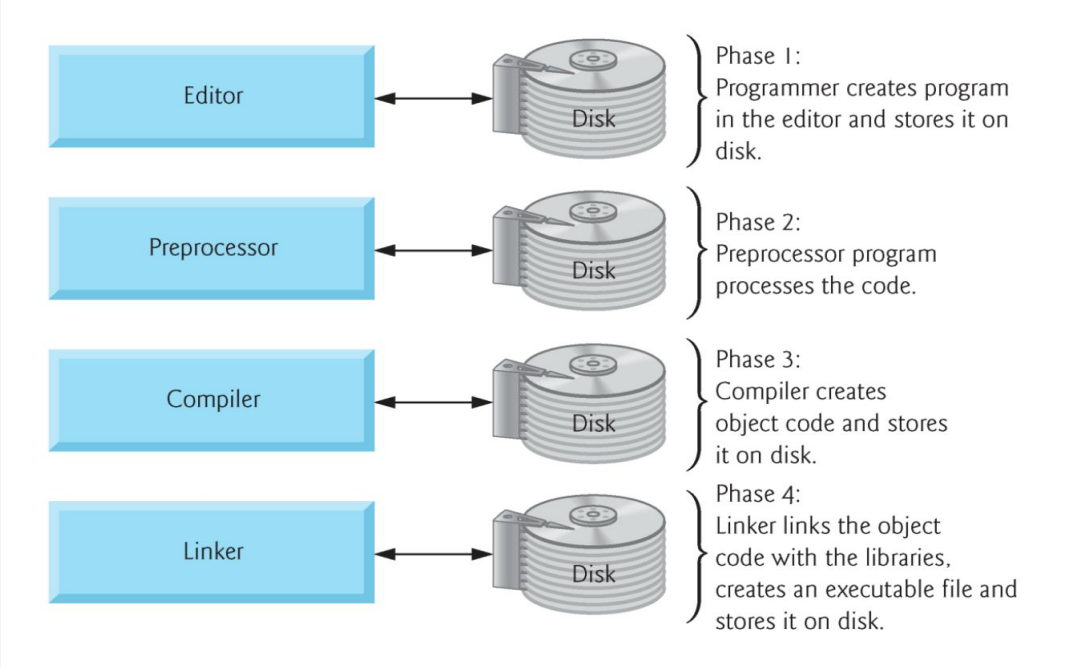


Figure 5: C development environment

Basic Structure of a C Program

- `#include <stdio.h>` – Standard I/O library
- `int main()` – Entry point of every C program
- `printf()` – Prints output to the screen
- `return 0` – Indicates successful execution

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}

```

Figure 6: Sample C program

Compiling, Linking, and Execution

To compile and run a C program, you need:

- A text editor: VS Code, Vim, Nano, or any IDE.
- A compiler: `gcc` (GNU Compiler Collection), Clang.
- A terminal: sh, bash, or any other terminal.

Compiling, Linking, and Execution

- **Preprocessing** – (`#include` macros are replaced)
- **Compilation** – (C code → Assembly code)
- **Assembly** – (Assembly code → Machine code)
- **Linking** – Combines multiple object files into a single binary executable file.
- Example command:

```
$ gcc -c main.c -o main.o  # Compiles to object file (no linking yet)
$ gcc main.o -o main  # Links and creates the final binary executable
```

Makefile

- Makefiles automate compilation and builds so you do not have to run multiple `gcc` commands manually.
- Example command:

```
$ make       # Compiles the program
$ make clean # Cleans up compiled files
```

make

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99

all: hello

hello: hello.o
    $(CC) $(CFLAGS) -o hello hello.o

hello.o: hello.c
    $(CC) $(CFLAGS) -c hello.c

clean:
    rm -f hello hello.o
```

Figure 7: Makefile example

Demo

- Basic usage of bash in LinuxPool machines.
- `clone`
- Writing a simple C program (hello.c). Compiling with gcc and running the executable.
- Using a Makefile.
- `commit`, `push`, and `pull`.